Lecture 16:

Last time

We saw a finer point or two on how functions are evaluated in R; we learned a bit about environments and found some interesting effects when functions return functions.

We also discussed an example of how the object-based dispatch methods in R work; it’s a slightly old example (relative to the data you’ve been looking at in this class so far) but a rich one.

Today

We will consider performance of R and debugging tools; these tend not to be very sexy but are important for helping to produce efficient code.

Once we have happy, efficient code, we might want to share it; and that is our second topic today.

Tools

Debugging is, in general, a cumbersome and tiring task. The debugging skill of the programmer is probably the biggest factor in the ability to debug a problem, but the difficulty of software debugging varies greatly with the programming language used and the available tools, such as debuggers. Debuggers are software tools which enable the programmer to monitor the execution of a program, step it, re-start it, run it in slow motion, change values in memory and even, in some cases, go back in time. The term debugger can also refer to the person who is doing the debugging.

Generally, high-level programming languages, such as Java, make debugging easier; because they have features such as exception handling that make real sources of erratic behaviour easier to spot. In lower-level programming languages such as C or assembly, bugs may cause silent problems such as memory corruption, and it is often difficult to see where the initial problem happened; in those cases, sophisticated debugging tools may be needed.

For debugging electronic hardware (e.g., computer hardware) as well as low-level software (e.g., BIOSes, device drivers) and firmware, instruments such as oscilloscopes, logic analyzers or in-circuit emulators (ICEs) are often used, alone or in combination. An ICE may perform many of the typical software debugger's tasks on low-level software and firmware.

Basic steps

Although each debugging experience is unique, certain general principles can be applied in debugging. This section particularly addresses debugging software, although many of these principles can also be applied to debugging hardware.

The basic steps in debugging are:

- Recognize that a bug exists
- Isolate the source of the bug
- Identify the cause of the bug
- Determine a fix for the bug
- Apply the fix and test it

Taken from: http://en.wikipedia.org/wiki/Debugging
Identifying a problem

We’ve already seen some simple tools to help identify problems; R includes functions `stop()` and `warning()` to handle “exceptions” as they are encountered.

While `stop()` always halts the execution and returns; `warning()` will behave differently depending on the value of the “option” `warn`.

If `warn==0`, warnings are ignored; if `warn==0`, they are stored up and printed after execution has completed; if `warn==1`, they are printed as they happen; and if `warn>=2`, R will turn warnings into errors.

An aside on options

Through the command `options()` you can examine and change a variety of global parameters which affect the way R computes, displays (prints) its results.

Similarly, graphical parameters are examined and changed with the function `par()`.

Back to debugging

R offers a simple browser that lets you explore code while it is running; once we have identified that there's a problem, we might want to go in and examine things more closely.

Once in the browser, you can type any R command; a few of them are treated differently, however, to control the browser's functioning.
You can also monitor a running program with the use of the trace facility; \texttt{trace(fun)} will cause R to printout the call to the function \texttt{fun} each time it is evaluated.

Once the bug is fixed, you can turn off the trace mechanism with the function call \texttt{untrace(fun)}.
Performance considerations

Generating bug-free code is often only one step in a larger programming process; we also want to ensure our code runs quickly (um, relative to our patience) and efficiently (um, relative to the capabilities of the machine).

So far, we have probably managed to complete our computations without much difficulty; our assignments have involved significantly reducing the data outside of R and executing relatively simple functions inside R.

Timing

R provides facilities for timing your code; the function `system.time(expr)` executes the indicated expression and records five kinds of timing information.

In a vector of length 5, R returns the user CPU time, the system CPU time*, the elapsed time (wall clock), and two times that measure child processes and are usually zero.

Let's consider a question from a previous class...

* System time measures low-level operations such as system calls, paging, and I/O that the operating systems does on behalf of the process.

Space

Notice that in R we often don't explicitly allocate space for an object; although in the previous example we might have some performance gains if we do.

Recall that objects in R translate into bits of your computer's main memory; if you have lots of objects in your workspace, your R process will take up more room.

We have already seen how one can list objects with `ls()` or remove objects with `rm()`; we can tell their size with `object.size()`.
Space

You can use `object.size()` to give you a rough sense of how much room you will need for your computation.

R also does behind-the-scenes garbage collection; here we define garbage to be memory that is associated with objects and computations that are no longer being used.

You can see R performing this operation with the function `gcinfo(TRUE)`; and you can turn it off with `gcinfo(FALSE)`.

Sharing

Now that you have your code running efficiently with a very tiny footprint you might want to share it with others.

One of R’s great innovations was the ability to easily publish/share/distribute R code.

This sharing is done in the form of packages...
Packages in R

R is extended through a series of packages; fruits of the programming labors of many, many others (um, mainly men, but we can change that!)

You can see which packages have been installed with an empty call to library()

Packages in library '/Library/Frameworks/R.framework/Resources/library':

- MASS: Main Package of Venables and Ripley's MASS
- base: The R Base Package
- boot: Bootstrap R (S-Plus) Functions (Canty)
- chron: Chronological objects which can handle dates and times
- class: Functions for classification
- cluster: Functions for clustering (by Kaufman et al.)
- datasets: The R Datasets Package
- foreign: Read data stored by Minitab, S, SAS, SPSS, Stata...
- grDevices: The R Graphics Devices and Support for Colours and Fonts
- graphics: The R Graphics Package
- grid: The Grid Graphics Package
- lattice: Lattice Graphics
- methods: Formal Methods and Classes
- mgcv: GAMs with GCV smoothness estimation and GAMMs by REML/PQL
- nlme: Linear and nonlinear mixed effects models
- nnet: Feed-forward Neural Networks and Multinomial Log-Linear Models
- rpart: Recursive Partitioning
- sna: Tools for Social Network Analysis
- spatial: Tools for Kriging and Point Pattern Analysis
- splines: Regression spline functions and classes
- stats: The R Stats Package
- stats4: Statistical functions using S4 classes
- survival: Survival analysis, including penalized likelihoods
- tools: Tools for Package Development
- utils: The R Utils Package

Packages in R

Each of these packages has been installed so that everyone calling R on a statistics department computer can use them

You can also install locally in your own R library
Packages in R

Technically, a package in R is a subdirectory containing a file DESCRIPTION and the subdirectories R, data, demo, inst, man, src, tests (some of which may be missing)

The subdirectory can also contain files INDEX, install.R, NAMESPACE, configure, cleanup and COPYING

Other files can be included that might help an end-user, but are ignored by R

The DESCRIPTION file

This file contains basic information about the package

Package: chron
Version: 2.2-33
Date: 2004-11-06
Author: S original by David James <dj@research.bell-labs.com>, R port by Kurt Hornik <Kurt.Hornik@R-project.org>.
Maintainer: Kurt Hornik <Kurt.Hornik@R-project.org>
Description: Chronological objects which can handle dates and times
Title: Chronological objects which can handle dates and times
Depends: R (>= 1.6.0)
License: GPL
Packaged: Sat Nov 6 09:06:09 2004; hornik
Built: R 2.0.0; powerpc-apple-darwin6.8; 2004-11-17 11:44:35; unix

Package: spatial
Description: Functions for kriging and point pattern analysis.
Title: Functions for Kriging and Point Pattern Analysis
Bundle: VR
Priority: recommended
Version: 7.2-8
Date: 2004-09-09
Depends: R (>= 2.0.0), graphics, stats
Suggests: lattice, nlme, survival
Author: S original by Venables & Ripley. R port by Brian Ripley <ripley@stats.ox.ac.uk>, following earlier work by Kurt Hornik and Albrecht Gebhardt.
Maintainer: Brian Ripley <ripley@stats.ox.ac.uk>
BundleDescription: Functions and datasets to support Venables and Ripley, 'Modern Applied Statistics with S' (4th edition).
License: GPL (version 2 or later) See file LICENCE.
URL: http://www.stats.ox.ac.uk/pub/MASS4/
Packaged: Thu Sep 9 21:11:28 2004; ripley
Built: R 2.0.0; powerpc-apple-darwin6.8; 2004-10-14 12:30:40; unix
The DESCRIPTION file

The fields Package, Version, License, Description, Title, Author and Maintainer are mandatory

These provide the basic information about the package and what a user is allowed to do with it

R provides a number of facilities for authoring packages, most of which are documented in "Writing R Extensions" on the CRAN web site

So that's the mechanism, but...

Let's reconsider the wafer software from last time; we have used the package mechanism to share our code with engineers in the company...

A couple years go by, the company sells its interest in semiconductor manufacturing (in 2002, Lucent sells off Agere)

A little while after that you get a call from a graduate student at a university who wants to see if they can get hold of the package to do their research

A couple of weeks go by and you get another call, this time from a researcher at Intel who wants to use the code

Finally, after another couple of weeks, you get a call from a company in silicon valley that makes testing equipment

Sharing code

Aside from the technical mechanism in place for sharing code are there other implications?

Maybe that eager graduate student will find and fix a few bugs, or maybe she'll even extend what we've done and add new analysis

Maybe Intel will create a GUI for our toolset and start to distribute it internally to its engineers

Or, maybe, the startup does a little market research and realizes that there's a huge need for tools of this kind and wants to sell it

What laws exist to govern these kinds of developments?

Copyright

Technically, any creative works you produce are automatically copyrighted; this includes software

This means you have the following rights

1. The exclusive right to copy the work;
2. The exclusive right to make derivative works;
3. The exclusive right to distribute the work;
4. The exclusive right to perform the work; and
5. The exclusive right to display the work

http://www.groklaw.net/article.php?story=20031231092027900
The Open Source software movement has created licenses, legal mechanisms, which allow people to use copyrighted material subject to certain restrictions. Typically, these licenses restrict how people can use, copy and redistribute software.

R is licenses under the GNU General Public License; It is the most popular Open Source software license:

1. The right to copy and redistribute so long as you include a copyright notice and a disclaimer of warranties. You may change for the cost of distribution and you may offer warranty protection for a fee.
2. The right to make derivative works for your own use.
3. The right to distribute derivative works so long as you:
   1. Identify the work as modified;
   2. License it under the GPL, and
   3. Provide the license information interactively if the program normally runs interactively.

This section, and the obligation to license under the GPL, does not apply to works which are independent works distributed with the GPL'd work and which run on the GPL'd works.

4. You may distribute the work only in executable form so long as the source code is:
   1. distributed with the object code;
   2. offered by a written offer, valid for a period of at least three years, to make the source code available for no more than the cost of distribution; and
   3. for non-commercial distributions, accompanied with the offer the redistributing party received as to the availability of the source code.

5. You may not impose restrictions on any of these rights.

http://www.groklaw.net/article.php?story=20031231092027900

Basically, you get rights to use, copy and redistribute provided you do so under the GPL license. This means that software can't become less "free" or "open" through successive enhancements.

License: Unlimited use and distribution (see LICENCE).
License: GPL (version 2 or later) See file LICENCE.
License: GPL Version 2 or later.
License: Unlimited distribution.
License: GPL
License: GPL (version 2 or later) See file LICENCE.
License: GPL version 2 or later
License: GPL Version 2 or later.
License: GPL Version 2 or later.
License: GPL Version 2 or later.
License: GPL Version 2 or later.
License: GPL Version 2 or later.
License: GPL Version 2 or later.
License: GPL Version 2 or later.
License: GPL (version 2 or later) See file LICENCE.
The typical line of reasoning...

If my software is disposable, who cares? If it’s worth something, why should I share it?

Ugh, the legalese is way too much to think about; Why not just let people use my software if they want?

Well...

The copyright is yours whether you like it or not; and placing something in the public domain is not entirely trivial

By attaching a license to a piece of software, you are saying that the software IS valuable, and that the value is enhanced by letting others participate in its evolution

The strength of the Open Source effort is that the value of software increases when a community of people contribute

And it’s not just software

A couple more months go by and you find yourself at a university creating a whole lot of educational materials

Again, you own the copyright; wouldn’t it be better to share?

Creative Commons builds off the work in the Open Source software effort and has designed a set of licenses that can be custom assembled to let you control how others can use your work (http://www.creativecommons.org)

GNU General Public License, Free Software Foundation

The GNU General Public License is a Free Software license. Like any Free Software license, it grants to you the four following freedoms:

- The freedom to run the program for any purpose.
- The freedom to study how the program works and adapt it to your needs.
- The freedom to redistribute copies so you can help your neighbor.
- The freedom to improve the program and release your improvements to the public, so that the whole community benefits.

You may exercise the freedoms specified here provided that you comply with the express conditions of this license. The principal conditions are:

You must conspicuously and appropriately publish on each copy distributed an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of the GNU General Public License along with the Program. Any translation of the GNU General Public License must be accompanied by the GNU General Public License.

If you modify your copy or copies of the Program or any portion of it, or if you develop a program based upon it, you must distribute the resulting work provided you do so under the GNU General Public License. Any translation of the GNU General Public License must be accompanied by the GNU General Public License.

If you copy or distribute the program, you must accompany it with the complete corresponding machine-readable source code or with a written offer, valid for at least three years, to furnish the complete corresponding machine-readable source code.
Next time

We will have a look at a DBMS; we will see how to interact with a database from R

We will probably also talk a bit about XML...